Bus (simple event bus, where we publish “actions” (“actions”?))

```
class Bus:
    def publish(self, topic: str, payload: dict | None = None) -> None
    def subscribe(self, topic: str, callback) -> None
    def unsubscribe(self, topic: str, callback) -> None
```

Router (maps NLU intent to a “skill” to execute)

```
class Router:
    async def start(self) -> None
    async def stop(self) -> None
    def add_route(self, pattern: str, skill_name: str) -> None
    async def route(self, intent: dict) -> None
```

TTS

```
class TTS:
    async def start(self) -> None
    async def stop(self) -> None
    async def synthesize(self, text: str, voice: str | None = None) -> str  # returns wav_path
```

Playback

```
class Playback:
    async def start(self) -> None
    async def stop(self) -> None
    async def play(self, wav_path: str, blocking: bool = False) -> None
    # publishes: audio.playback.start, audio.envelope (optional), audio.playback.end
```

StepperMotorDriver (exists)

Drives a stepper for the mouth (or other axes)

```
class StepperMotorDriver:
```

```
    # Lifecycle / board setup
    def open(self) -> bool
    def close(self) -> None
    def reset(self) -> None

    # Configuration
    def attach_pins(self, name: str, step_pin: int, dir_pin: int,
                    en_pin: int | None = None, ms_pins: tuple[int, int, int] | None = None) -> None
    def set_max_speed(self, name: str, steps_per_s: float) -> None
    def set_accel(self, name: str, steps_per_s2: float) -> None

    # Enable/disable
    def enable(self, name: str, enabled: bool) -> None
    def is_enabled(self, name: str) -> bool

    # Motion
    def move_to(self, name: str, target_steps: int) -> None       # non-blocking
    def stop(self, name: str, hard: bool = False) -> None
```

MouthController (layer over motor driver)

Maps 0..1 mouth openness to step positions.

```
class MouthController:
    def __init__(self, stepper: StepperMotorDriver):
        ...

    def initialize_hardware(self) -> bool
    def shutdown_hardware(self) -> None


    def set_mouth_position(self, position: float) -> None

    def animate_mouth(self, envelope: list[float], fps: int = 60) -> None
    def idle_pose(self) -> None
    def safe_pose(self) -> None  # stop motion, mute amp, move to safe steps
```

MouthAnimator (lip-sync, will use MouthController. May combine these)

```
class MouthAnimator:
```

```
    def configure(self, min_open: float, max_open: float,
                  attack_ms: int, release_ms: int, fps: int = 60) -> None
    def attach(self, hardware: HardwareController) -> None

    # Event hooks
    def on_audio_play_start(self, meta: dict) -> None
    def on_audio_frame_envelope(self, t_ms: int, env: float) -> None  # 0..1
    def on_audio_play_end(self) -> None

    # Offline
    def render_envelope(self, wav_path: str, frame_ms: int = 16) -> list[float]
```

Microphone (exists)

Keeps STT decoupled from device details

```
class Microphone:
    def __init__(self):
        ...

    def start(self) -> None
    def stop(self) -> None
    def read_chunk(self) -> bytes
```

Speaker (exists)

```
class Speaker:
    def __init__(self):
        ...

    def start(self) -> None
    def stop(self) -> None
    def play_pcm(self, pcm: bytes) -> None
    def set_volume_db(self, vol_db: float) -> None
    def mute(self, muted: bool) -> None
```

STT

```
class STT:
    async def start(self) -> None
```

```
    async def stop(self) -> None
    def set_vad_params(self, threshold: float, min_ms: int, max_ms: int) -> None
    def on_audio_chunk(self, pcm: bytes, t_ms: int) -> None
    def partial_transcript(self) -> str | None
    # publishes "stt.transcript": {"text": str, "confidence": float}
```

NLU

```
class NLU:
    async def start(self) -> None
    async def stop(self) -> None
    def parse(self, text: str) -> dict  # {"intent": str, "entities": dict, "confidence": float}
```

SkillManager (Might split each skill up into its own class)

```
class SkillManager:
    async def start(self) -> None
    async def stop(self) -> None
    def register(self, name: str, skill_obj) -> None
    async def invoke(self, name: str, request: dict) -> dict  # e.g., {"say": "...", ...}
```

Main Software Flow: